1-1-2002

A genealogical approach to building a denial of service attack taxonomy

Gregory Warren Rice
Iowa State University

A genealogical approach to building a denial of service attack taxonomy

by

Gregory Warren Rice

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Jim Davis, Major Professor
Cliff Bergman
Doug Jacobson

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the master's thesis of

Gregory Warren Rice

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Throughout the entire research effort, I received dedicated support from professors, staff, and several students at the University. The flexibility, understanding, and guidance offered by Dr. Jim Davis, my major advisor, all have been outstanding. Much of his input and patience during both research and classroom work has been greatly appreciated.

In addition, I would like to sincerely thank both of my parents, Robert and Linda Rice. Their dedication towards my extracurricular activity involvement and education throughout much of my life has been greatly underappreciated.

Finally and foremost, I would like to thank my friend and partner, Cara Harris. Her support and motivation throughout much of the final portion of this project has been wonderful. The understanding and patience which she displayed over the past year has been exceptional. I simply could not ask for more.

ABSTRACT

Availability requires that computer systems remain functioning as expected without degradation in processing, access, or availability of resources to legitimate users. Although many organizations may have implemented good security practices in building their networks, these networks still remain open to common assault tools that threaten the availability of network services to legitimate users. Over time, many of these availability assaults, also known as denial of service (DoS) attacks, have grown more complex, effective, and even easier to launch.

Unfortunately, the number of published attacks continues to grow while few security researchers firmly understand their details. If properly compiled into an effective database, the collection of these different attack scripts could possibly provide valuable information to computer security engineers such as characterizing threats in terms of source, attack method, and effects on computer resources. Using the attack database, it is also possible to begin to build a taxonomy of common denial of service attacks and develop a general methodology for describing and characterizing such threats. Although various research studies have been previously conducted in hopes of building a general software vulnerability database for use by security analysts, no studies have specifically focused on studying attack histories.

By examining DoS attack history, genealogy, and taxonomy together, researchers gain the ability to not only identify existing attacks and possible countermeasures but possibly even predict future attacks in some cases as well. Although attacks have grown increasingly complex over time, many of the same basic ideas and methods for performing the denial of service remain unchanged or only slightly modified. While previous research models had focused on attacks as singular data points, modeling assaults as growing genealogical trees formed from several different software attacks yields valuable information on recurring themes in DoS attacks. Furthermore, attack tree hierarchies allow researchers the ability to study how software vulnerability exploits have changed over time. Building a vulnerability database of denial of service attacks comprised of both singular entries and corresponding attack trees allows for the development of classifications in the taxonomies of vulnerabilities and reveals characteristics of attacks that have remained prevalent in software over time.

INTRODUCTION

As the connectivity of computers continues to grow at enormous rates and the functionality of computer networks continues to expand, users are beginning to increasingly rely on the communication infrastructure. Through the use of computer networks, users access computer resources from remote office locations, citizens access news across the globe, and others share the ability to interact with new people all online. While maintaining the ability to reliably transmit and receive data across these computer networks has long been emphasized by designers, achieving these goals securely is perhaps a more recent focus.

Secure computers are those computing systems which can be depended upon and behave as expected to [1]. In other words, the user expects to be able to reliably interact with software on his or her system and transmit and receive data with other sources. However, some of the main issues of concern in any data communication network is confidentiality, privacy, and availability [2]. In order to achieve confidentiality, data must be protected such that only those authorized and legitimate users have the ability to access it. Likewise, achieving data integrity requires that information be protected from being altered in any way either by unauthorized user or by accident. Over the last few years, many organizations have expended considerable resources to protect internal infrastructure from such external compromise [3]. However, maintaining availability of computer and network resources still remains as an even more challenging task.

Problem Statement

Availability requires that computer systems remain functioning as expected without degradation in processing, access, or availability of resources to legitimate users. Although many organizations may have implemented well planned and good security practices in building their networks, these networks still remain open to common availability attacks. In other words, traditionally well-protected infrastructures all suffer from vulnerabilities in maintaining availability. In fact the existence of previous security practices and management plans may indeed actually make the network more vulnerable to attack due to the false pretense of immunity.

Many of these assaults, otherwise known as denial of service (DoS) attacks, are commonly published on a wide variety of Internet and security web sites. In fact many sites freely advertise and distribute easy-to-use attack scripts. Web-surfers, who may know little to nothing of the technical details of the attack, can easily download the scripts and launch large bombardments against computer networks and computer systems. As a result, many attacks are easy to carry out and may even occur on a relatively frequent basis, simply going unreported. Furthermore, these assaults have grown increasingly more complex and effective over the past few years. With an increased understanding of how systems work, intruders have become skilled at determining weaknesses in systems and exploiting them [4]. As new security measures are implemented, new attacks begin to appear.

If properly compiled into an effective database, the large collection of attack scripts widely available on the Internet could possibly provide valuable information to computer security researchers. The collection, commonly referred to as the Global Attack Toolkit (GAT), could be used to generate statistics on important characteristics of attacks, such as what percentage of attacks are launchable from a Windows host, what percentage are remote penetration attacks, or what percentage use common communication protocols such as common TCP [5]. By composing a threat database, researchers have the ability to specifically analyze how attacks occur and model how such attacks can be prevented. Furthermore, by characterizing threats in terms of source, attack method, and effects on computer resources, researchers have the ability to select common high-level threats and set objectives for responding to attacks.

Thesis Statement

Using the attack database, it is also possible to begin to build a taxonomy of common denial of service attacks and develop a general methodology for describing and characterizing such threats. Currently, several hundred denial of service attacks exist, over 300 of which have been catalogued here. However, little is known about these attacks other than the fact that they exist [5]. By classifying existing threats, researchers gain the ability to understand the structure behind DoS attacks and how such threats have changed in functionality over the past several years. More importantly though, security engineers can

examine attacks for common instances that different attack denial of service tools and scripts may share, how different assaults relate, and how attacks have changed over time.

Although various research studies have been previously conducted in hopes of building a general software vulnerability database for use by security engineers, analysts, and law enforcement officials, no studies have specifically focused on studying attack histories. Research conducted by Ivan Krsul [6], Peter Mell [5], Tom Richardson [13], and others have all focused on compiling vulnerabilities into databases and analyzing the entries for common characteristics in aim of building attack classification taxonomies. For example Krsul describes his software vulnerability analysis as "a framework for the development of taxonomies according to generally accepted principles [that] can be used to develop unambiguous classifications" [6]. By characterizing existing threats and increasing our understanding of the nature of software vulnerabilities, software developers can hopefully improve the design of products to withstand such attacks in the future.

However, unlike those previous research projects aimed primarily at developing only attack and vulnerability taxonomies, the purpose of this research project was not only to continue to refine denial of service attack taxonomies, but more importantly to study how denial of service tools have changed over time. By examining DoS attack history and taxonomy together, researchers gain the ability to not only identify existing attacks and possible countermeasures but possibly even predict future attacks in some cases as well. Although attacks have grown increasingly complex over time, many of the same basic ideas and methods for performing the denial of service remain unchanged or only slightly modified. Building a taxonomy of Internet denial of service threats avoids confusion between researchers and coordinates development efforts of security mechanisms. While previous research models had focused on denial of service attacks as singular data points, modeling attacks as growing genealogical trees formed from several different software attacks yields valuable information on recurring themes in DoS attacks. Furthermore, attack tree hierarchies allow researchers the ability to study how software vulnerability exploits have changed, migrated, and increased in complexity over time. Building a vulnerability database of denial of service attacks comprised of both singular entries and corresponding

attack trees allows for the development of classifications in the taxonomies of vulnerabilities and reveals characteristics of attacks that have remained prevalent in software over time.

Thesis Organization

The remainder of the research presentation is organized to give the reader a firm background in the denial of service attacks and previous research as well as present the new ideas formulated by the thesis. Chapter 2 thoroughly establishes the mechanisms of denial of service attack as well as expanding on research conducted by others. Furthermore, Chapter 3 establishes the database structure and hierarchy used to conduct the thesis research. Chapter 4 details many of the past trends noted in the attack taxonomy as well as the recurring themes within the specific categories of the taxonomy. By the same token, Chapter 5 discusses several important recurring themes in the genealogical attack trees, while Chapter 6 notes important countermeasure steps to prepare for such assaults and makes recommendations on future work. Finally, Chapter 7 presents a general conclusion to the research study and thesis.

BACKGROUND

In February of 2000, a series of massive availability attacks incapacitated several high-profile Internet sites, including Yahoo, Ebay, and E*trade. Next, in January of 2001, Microsoft's name server infrastructure was disabled by a similar assault [7]. Since September 1996, several dozen sites on the Internet have been subjected to a denial of service attack [8]. The attacks can be launched with little effort and are often difficult to trace back to the originator. Furthermore, it is never necessary for the attacker to explicitly break any security mechanism in place to exploit the vulnerability. Complex and strong authentication mechanisms that provide data confidentiality and integrity are often rendered useless in availability attacks since the enemy never actually needs to compromise the computer system security.

Definition

The primary goal of an availability attack is simply to deny victims access to a particular computer resource. The enemy never needs to supply a password, secure token, or other means of authentication; the attacker only needs to block other users from accessing the system. A denial of service attack is characterized by an explicit attempt by attackers to prevent users of a service from using that service [9]. For example, flooding a network with traffic prevents other computing systems from communicating on the same network and thereby disrupts normal network services. In this scenario the attacker never needs to authenticate with any computer system. They simply overload normal resources and effectively block communication.

Note that not all availability attacks are necessarily denial of service attacks. Many other attacks may include a denial of service as a component [9]. Often an availability attack is simply an indirect result of a larger attack goal. For example, an attacker exploiting a software vulnerability on a web server may exhaust all memory resources and deny other web surfers access, but they may be doing so with the overall purpose of gaining system privileges to the server. Likewise illegitimate use of computing resources may also result in denial of service [9]. For instance, users who set up anonymous FTP servers on public computing systems often subject the server to such intense traffic that disk space and CPU

resources are quickly exhausted. Furthermore, DoS attacks can target any computing system including application software, operating system, hardware, network devices, and the network protocol itself.

Denial of service attacks can disable any computer from performing normal system routines or accessing data across network resources. Furthermore, launching many denial of service attacks often require little in the way of computing resources, even against large and sophisticated servers. Given the goals of denial of service attacks, exploitation of the vulnerabilities typically occur by taking advantage of process recovery, crashing a particular process in the information flow, or overloading a system resource such that information flows in a very untimely fashion.

Designed Outage

Many computing devices commonly possess some means of process recovery such as a system reset function or manual override. These functions perform a temporary shutdown and reload of all necessary services for the device. Likewise these functions are included by engineers as part of the system by design in order to maintain steady state. For example, any TCP session requires a sender and receiver address as well as a sender and receiver port. At the end of the session, the system closes the connection by not only closing the current session but also by prohibiting any future connections on those particular port addresses for a specific period of time. Here the goal is to perform the steady state of communication on the system by reducing conflicts between simultaneous and later connections. The overall purpose of designed outage is reduced system errors and error recovery.

However, denial of service attacks can take an unfair advantage over designed outage functions by using them maliciously [10]. For example, malicious code could call a process to hang or reset a system. Likewise a more common denial of service attack is IP spoofing and blocking. If an attacker assumes the IP address of a receiving system and sends multiple forged data packets on the receiving system's or victim's behalf, he or she has the ability to drop the communication session between the sending and receiving systems, thereby denying legitimate communication service.

Resource Destruction

Given the complexity of modern software packages, hundreds of known and unknown flaws exist in release versions of commercial programs. Resource destruction occurs when a program crashes [10]. By exploiting uncovered flaws in software programs, attackers have the ability trigger system failures. For instance, Hewlett Packard manufactures network hardware, commonly referred to as JetDirects, which enable network printer sharing of standard printers with only parallel interfaces. However, a flaw discovered in the JetDirect software demonstrated a vulnerability where any user connecting to the device with an FTP client could effectively cause a complete device failure. In fact the JetDirect would require a complete reset before functioning properly.

Attackers have the ability to remotely force destructive states in software simply by exploiting vulnerabilities and flaws in system software and hardware. Likewise in some cases, intruders can manipulate configuration information in order to prevent a legitimate user from accessing resources. For example, if an intruder can change the routing information on the network devices, they may be able to effectively disable network communication. In other cases, attackers who modify the registry on a Windows machine may gain the ability to disable certain system functions [9]. In extreme cases, resource destruction can even consist of the physical destruction of computing resources.

Resource Exhaustion

Even in cases of well designed software and proven security algorithms, resource exhaustion attacks can effectively deny service without ever exploiting software vulnerabilities or compromising system security. In these attack scenarios, attackers rapidly consume scarce, limited, or non-renewable computing system or network resources. DoS attacks are most frequently executed against network connectivity, where the goal is to prevent hosts from communicating on the network by exhausting the systems resources [11]. Any process that requests more of a memory or CPU resource can then be denied simply by consuming all available resources on a system.

For example a common resource exhaustion threat prevalently exploited by attackers is the SYN-Flood attack. During the assault, attackers simply exhaust memory and CPU

resources of the victim system through opening several bogus connections with the server. According to the TCP communication protocol, a communication is established through a series of three packet exchanges, commonly referred to as a three-way handshake. Typically a connection request is made by the client to the server. The server then responds with an acknowledgement to the client, who then follows with the final packet in the three-way handshake exchange. An attacker initiates a SYN flood attack by sending many connections requests with spoofed addresses to the victim machine [12]. Each time a connection request is made, the server allocates system resources to the client request and respond with an acknowledgement. However, in this case the client never responds with any additional data. Instead the attacker only responds with several more connection requests until eventually the server reaches a limit on half-open connections or exhausts all available system resources. Due to the lack of authentication in the TCP/IP protocol, attackers have the ability to carry out attacks that place undue burdens on system computing resources.

In other cases, attackers may actually use a system's own computing resources against it in order to deny service. For example, in the Chargen attack the intruder uses forged UDP packets to connect the echo service on one machine to the chargen service on another machine [11]. The two services interact continuously such that all available network bandwidth is consumed completely between the two machines. Thus, the network connectivity for all machines on the same networks as either of the targeted machines may be affected [11]. By utilizing two simple and legitimate services on established computing machines, attackers can force the machines to begin performing unexpectedly, creating adverse effects for others.

Although launching attacks from a remote location which consume all available network bandwidth may be difficult to do from a single client machine today, the vulnerability of network availability attacks still exist. By simply generating a large number of data frames on the computer network, attackers have the ability to maximize the current bandwidth of any network. Typically, these packets are ICMP ECHO packets, but in principle they may be anything [9]. In fact, new attacks exist which coordinate distributed functions through hordes of controlled machines to exhaust network resources. Modern

attacks coordinate the packet generation through several machines on different networks in order to overload the victim's bandwidth and effectively deny service.

Likewise, in addition to consuming large amounts of network bandwidth, malicious users can also consume other system resources as well. As previously mentioned the SYN-Flood attack exhausts resources that prevent network connectivity. Likewise, intruders can also consume other normal computing system resources as well. For example, email solicitation to all users on a particular server rapidly generates an excessive number of messages, thereby consuming a large quantity of disk space.

Moreover, many times attackers will use the same security mechanisms already implemented on a network in order to effectively cause resource exhaustion attacks. For example, intrusion detection systems examine current network traffic for commonly known attack exploits. Once an attack exploit is identified error messages are displayed and logged with the system administrator. Once again by generating bogus attack traffic, an intruder could possibly slow intrusion detection systems and exhaust disk resources. Likewise, many sites have scheme in place to lockout an account after a failed number of login attempts [9]. After a user consistently enters their password incorrectly, within a given time period, the account is disabled. While often such lockouts are a good security practice to prevent intruders from guessing user passwords, the same security mechanism can be used to launch an attack where the enemy enters wrong passwords on behalf of the user thereby disabling their account.

DEVELOPMENT OF A TAXONOMY

The focus of the research was on remotely exploitable attacks. In other words, only attacks which disable availability of computer resources to legitimate users by attacking some vulnerability from a remote network location were considered as applicable. The intent was to build a database of such attacks, sort attacks into specific categories, and most importantly study how the attacks changed over time. Previous research noted limitations of vulnerability databases in that all database entries were listed as being singular. When an exploit is posted on one of the online sites, it ends up being just one entry into the database; there is no indication of how popular that form attack is and how often it may be used in the subsequent days, weeks, and months after its posting [13]. In other words, by studying the temporal data of relationships between attacks and attack history, it is possible to model how attacks actively changed over time.

Database Structure

Often after a software vulnerability is discovered and published, exploitation attacks are developed by either malicious programmers or security engineers to demonstrate the weakness. Over time these attacks migrate and spread throughout different Internet security groups, individuals, and websites. Consequently, the attacks are often modified by other programmers to increase effectiveness, expand the attack to different software platforms, or even incorporate other denial of service models. As software vendors release fixes that repair vulnerabilities uncovered by the attack exploits, tools are also subsequently replaced or changed as new techniques are discovered to exploit other weaknesses.

Thus the vulnerability research database had a dual purpose of providing information on the attack as well as information on the attack taxonomy structure and relationship to other attacks. Important vulnerabilities matching the remote exploitation and denial of service criteria that were reported between 1997 and 2001 were compiled in the database along with information pertaining to attack publication date, software attacker platform, software victim platform, attack categorization type, and attack details. In addition, attacks were specifically studied and compared to other previous and existing attacks to help build historical and genealogical hierarchies of software vulnerability exploits. Previous taxonomy

research performed by Richardson, Krsul, and others was of little help here. As mentioned before, existing software vulnerability databases contained only singular entries. Instead of simply compiling data on software vulnerability details, attack scripts and exploit source codes were specifically analyzed and compared to build genealogical hierarchies and predict future denial of service tools.

Database Categories

Building an attack taxonomy necessitates the need for attack categories to simplify database design and research. Rather than rely on the vulnerability types previously discussed, several more descriptive categories were implemented in characterizing attacks. These categories, derived from the previous work of Richardson, include specification weakness, implementation weakness, and brute force attacks. Of those attacks, both brute force and specification weakness attacks are perhaps the hardest to overcome. Each detail the inherent weaknesses in software design and protocol specification that directly threaten securing the availability of computing resources.

Specification Weakness

In specification weakness attacks, vulnerabilities often exist in the definition of a network protocol or software algorithm itself. Often, the attack performs a legitimate operation that may go unhandled by the network protocol, or perhaps the attack exploits a weakness in the protocol definition. For example, DNS spoofing is a malicious attack often performed to deny the appropriate user service. The attack is legitimate in the sense that the attacker is performing a legitimate DNS operation. However, in this case no strong authentication mechanism was built into the DNS and IP protocols that would prevent such an attack. Likewise, the recent publication of vulnerabilities in the WEP protocol outlines specific problems in protecting data confidentiality on wireless networks [14]. Here, a protocol design flaw in improperly implementing a shared key encryption scheme allows attackers to remotely spy on data communication. Hence incomplete or ambiguous protocol specification may lead to inadvertent security vulnerabilities.

Because the designers of protocols are often either not familiar with security or simply unconcerned with the issue, protocol specifications are typically susceptible to either authentication, modification, or, as in our case, availability attacks. For example, in the case of WEP design, engineers did not invite any security experts to take part in the design or certification of the privacy algorithm. As a result, the encryption scheme used to protect wireless networks today is susceptible to assault. Likewise, designers of the TCP protocol did not foresee security considerations during the design of the fragmentation specification. Several popular attacks exist which set the fragment size of a TCP communication interchange to abnormal sizes. Since the specification for implementation of the protocol does not specify fragment offset validation, it is often possible for an attacker to exhaust system resources on the vulnerable host simply by modifying the data pointers to unreasonable values.

For the purpose of this study, a specification weakness attack is any attempt by a malicious user to directly take advantage of a flaw, weakness, or security vulnerability in the design or specification of a network, service, or communication protocol. It is extremely important to note that specification weakness attacks do not include those attacks where the software or hardware designers utilizing the protocol incorrectly implement the specification. Instead this category includes those attacks where the protocol designers have incompletely, ambiguously, or mistakenly specified the implementation. Attacks which specifically exploit a flaw in the implementation of protocol specification fall into a separate category known as implementation weaknesses. Specification weakness attacks are only those malicious efforts which expressly focus on exploiting inherent flaws in the protocol design.

Brute Force

On the other hand, brute force attacks often do not rely on exploiting any specific software or protocol vulnerability. Instead, the attacks are once again completely legitimate according to the engineering design specifications. However, brute force attacks simply function by overwhelming the victim with the illegitimate use of resources in an entirely legitimate fashion. For instance, an attacker could overwhelm a web server with several hundred connection requests in a very short period of time. Such an attack quickly exhausts

the resources of the victim to a point were responses to other normal user requests can no longer be made. Brute force attacks are extremely dangerous given the fact that most assaults are extremely effective and no strong countermeasure currently exists.

Often it can be difficult to distinguish some brute force attacks from specification weakness attacks. For example, in the classic Squid attack, an assaulter can effectively deny service to legitimate users simply by exhausting system resources on the victim machine. Since TCP implementations are designed with a small limit on how many open connections per port are possible at any give time, an attacker may initiate a Squid attack by sending multiple connection requests to the victim machine in a short period of time. In this scenario, the protocol is indeed being exploited since no means of authentication has been implemented by the TCP designers. However, the attack is only effective as a brute force means of exhausting system resources. In order to be entirely successful, the attacker must demonstrate the ability to make connection requests faster than the victim can allocate system resources to those requests. In other words, the enemy needs to continuously keep requesting new connections from the victim machine for the length of the attack [12]. More importantly though, once the attacker ceases connection requests and the victim has time to respond to those invalid requests and free system resources, the overall attack ceases as well.

For the purpose of this research study, a brute force attack is any attempt made by a malicious user to overflow or deplete computing, memory, or network resources by some forceful and exhaustive means that relies on the attackers own processing power and network bandwidth and the limitations of the system and network resources of the victim machine. While other attacks maintain effectiveness until both the victim system is reset and the attacker ceases the denial of service attempt, brute force attacks often lose all effectiveness soon after the attacker stops efforts. Instead of relying on the exploitation of software and hardware vulnerabilities, brute force attacks simply overwhelm the victim's computer resources and deny service to other legitimate users.

Implementation Weakness

Finally implementation weakness attacks are perhaps the most prevalent. These attacks stem from specific software vulnerabilities that result from improper design

validation, poor engineering design, or even poor error recovery mechanisms. For example, popular buffer overflow attacks often exploit weaknesses in the software design that prevent improper data input by the user. Several attacks exist against Microsoft operating system software that allow malicious users to send improper data to the network protocol stack to cause the computing resource to deny service to other remote users. Because implementation weakness attacks exploit flaws in the software design itself, they are often fixed by vendors through the release of updated software packages and patches. Thus, implementation weakness attacks are perhaps some of the most important attacks to this study since many of the discovered attacks demonstrate migration and change over the research period.

Implementation weakness attacks are the result of mistakes made by the software and hardware engineers. Often computing products are rushed to market without proper design verification and testing. During the lifetime of the product, users, researchers, and possibly malicious attackers may discover small flaws in the system design, known as vulnerabilities. By exploiting the design implementation vulnerability, an attacker could possibly either read or modify system data, or, as in our case again, deny computing resources to other users. Buffer overflows, format strings, Unicode, and other attacks all fall into this category. For example, supplying an unusually long string of arbitrary characters in the username field of a Novell iManage server will result in page fault exception error that effectively denies all other users access to system resources until the server is reset. Thus the iManage denial of service attack is the result of a design mistake by failing to perform proper input validation and buffer checking.

More specifically though for the purpose of this research taxonomy, an implementation weakness attack is any exploitation of a hardware or software design flaw that directly threatens the availability of system resources. Unlike brute force attacks, implementation weakness attacks depend on neither the availability of resources by the attacker nor the exhaustion of resources of the victim. Instead implementation weakness attacks simply exploit vulnerabilities in the design implementation that lead to critical errors forcing a system halt, reset, or resource exhaustion.

Attack Trees

More important to refining a denial of service attack taxonomy, attacks were specifically studied for historical changes. In other words, how each assault, and DoS attack category for that matter, has changed over time with new technology, new software systems, and new security techniques. Instead of simply analyzing attacks for specific themes and characteristics and categorizing them accordingly, the primary research focus was on building attack trees, a hierarchical approach that attempts to trace the genealogy of DoS attacks and study modifications in their design. By modeling denial of service attacks as growing trees comprised of several different attack generations and possibly different attack categories, researchers gain the ability to study both how software and hardware vulnerabilities have changed over time as well as the exploitation of those vulnerabilities.

Building a vulnerability database of denial of service attacks comprised of not only singular categorical entries but growing attack tree hierarchies as well allows for the refinement of the taxonomy of vulnerabilities and reveals characteristics of denial of service attacks that have remained prevalent over time. Constructing attack graphs is a crucial part of doing vulnerability analysis of a network of hosts [15]. Because each path and node in the attack tree represents a slightly different means of achieving the desired denial of service, attack trees assess the overall availability vulnerabilities of a single network host. Developing attack trees which portray the genealogy of different denial of service attacks allows software and hardware designers to understand how both vulnerabilities and exploits have changed over time as well as concentrate on methods of predicting future attack patterns.

Together both the taxonomy of denial of service attacks coupled with their tree structures allows for refining the classification of existing vulnerabilities and understanding their growth and origin. Most importantly attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks [16]. In other words attack trees model the means of achieving the end goal of denial of service through slightly different means of approach. More formally defined,

> An attack graph or AG is a tuple $G = (S, \tau, S_0, S_s)$ where S is a set of states, $\tau \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, and $S_s \subseteq S$ is a set of success

> states. Intuitively S_s denotes the set of states where the intruder has achieved his goals. Unless otherwise stated we assume that the transition relation τ is total. We define an execution fragment as a finite sequence of states $s_0 s_1 \ldots s_n$ such that $(s_i, s_{i+1}) \in \tau$ for all $0 \leq i < n$. An execution fragment with $s_0 \in S_0$ is an execution, and an execution whose final state is in S_s is an attack, i.e., the execution corresponds to a sequence of atomic attacks leading to the intruder's goal state [15].

In other words attack graphs are data structures used to model all possible avenues of attacking a network. The graph offers a formal methodology for understanding the software vulnerability and how attacks exploit those vulnerabilities to achieve their goals.

More specific to this study however, attack trees are defined as the set of states that identify all possible means of using a particular vulnerability to achieve a denial of service. Attack trees are structures comprised of singular nodes, or exploits, that all have the desired goal of causing a loss of availability. Unlike the general attack graph definition, attack trees do not outline every possible means of achieving the end success state. Instead, the trees illustrate the evolution of a single attack over time. In other words, attack trees represent the single process the enemies designed to reach the goal state. Unlike their attack graph counterparts, attack trees only represent the genealogy of an attack, or how the attack has shown modifications and grown in complexity overtime.

Thus each attack tree begins with a root node of the goal state, a complete or partial denial of service, and one or more leaf nodes as means of achieving that desired goal. However each leaf node typically parented by another node which typically represents a more classical or conventional approach or a different means of conducting the same attack. For example, imagine that a criminal wishes to open a locked safe. While there may be several means of opening the safe through destruction, cutting, or explosives, the criminal wishes to open the locked safe through a single means of attack by cracking the combination. A wide variety of methods exist though for learning the combination of the safe as illustrated in Figure 1.

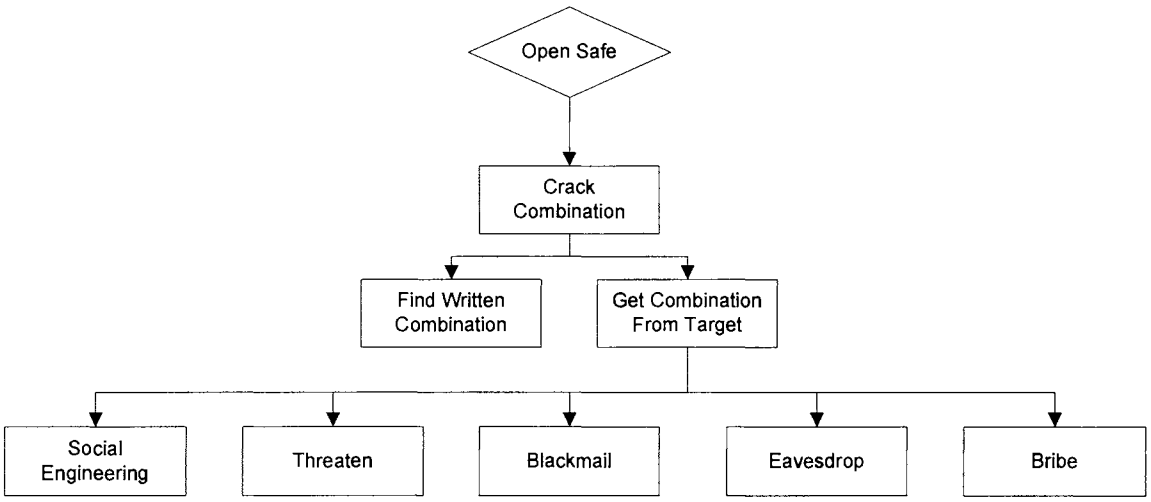

Figure 1. Safe Lock Attack Tree [16]

It is important to notice that several possible methods exist for conducting a single type of attack. The criminal can either steal the combination through eavesdrop or social engineering, or the attacker may possibly find the written combination by some other method. However, every child node stems from the parent attack method of cracking the combination.

Historically many of the means of cracking the safe can change as well. For example, the criminal may gain several new automated lock-picking kits as new technology is developed or changes are made to the safe design. As a result the tree may grow several new child nodes that stem either from previous child nodes or the parent node itself as demonstrated in Figure 2. Such historical and genealogical changes to the attack tree are extremely important to this research as they may demonstrate how vulnerabilities and attacks have changed over time with new software and hardware technology. Even in the modified attack tree where new means of conducting the same attack have been developed, each node becomes a subgoal, and children of that node are ways to achieve that subgoal [16]. More simply, attack trees simply illustrate every possible procedure for using a specific means to launch an attack.

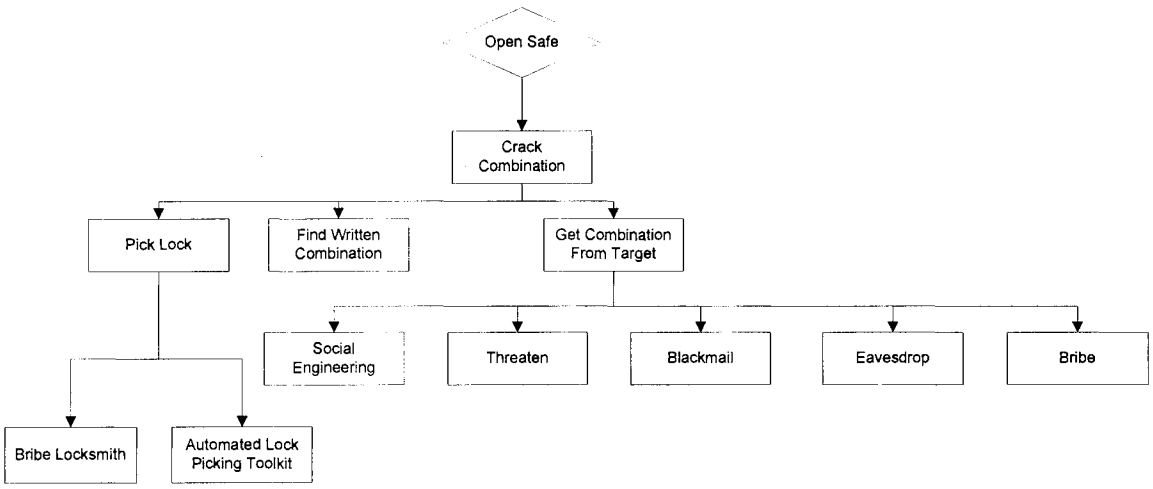

Figure 2. Modified Safe Lock Attack Tree

Unfortunately, developing a genealogical hierarchy of attack evolution is no simple task. Building the attack trees requires close examination of both the vulnerability description and exploit source code. However by building attack trees and studying their growth, evolution, and structure, researchers can determine not only likely methods of attack penetration but possibly predict future modifications to attacks as well. Much like security, attack trees are a process, or methodical method, of describing the vulnerabilities of any system, accounting for those which are most likely to occur, and repeating the entire process in a cycle fashion.

ATTACK DATABASE ANALYSIS

Since the early 1990's, thousands of different software vulnerabilities have been discovered. Every day, new software vulnerabilities are published in online databases and attack toolkits are developed to exploit the problems. During 1998 approximately 27% of these attacks were remote denial of service exploits [5]. In other words, many of the attacks tools under development are designed with the goal of blocking availability to legitimate users. However until taxonomies were developed for studying these attacks little was known of their characteristics and origins. Even when a host was attacked, it was difficult to predict how the attack occurred [5]. Building the attack database allows researchers to gain important statistics on what attacks exist, what they target, and what procedure they use for reaching the goal state.

Initially, the research effort focused on compiling data from Rootshell.com, Technotronic.com, Packet Storm Security, and Security Focus. Strong and effective remote denial of service attacks primarily on software systems were compiled in a Microsoft Access database in order to facilitate statistical analysis. When possible exploitation scripts were also captured to help facilitate the analysis of the attack origin, modifications, and genealogical growth. Overall, more than 300 attacks since 1996 were compiled as entries in the database. It should be noted however, that the database is not entirely comprehensive over the study. Several attacks published during the study period were of such trivial effectiveness that they had little significance as data points. Of prime importance to this study are attacks that are either entirely or at least partially successful in achieving the goal attack state.

Taxonomy Trends

As attack scripts, toolkits, and other exploit programs were entered in the database along with their respective vulnerability description each was then categorized according to the taxonomy structure described previously. Each denial of service technique was classified as being an implementation weakness, brute force, or specification weakness attack according to each category definition. Although a description of the vulnerability was included, it is once again important to note that only the attack is being classified. Unlike

vulnerability classification schemes, attack classification schemes are not necessarily concerned with identifying the specific exploited flaw [5].
In other words, the research effort primarily focused on cataloguing and script or set of instructions that a malicious user could execute in order to reach a state of denial of service on the victim system.

While some may argue that vulnerability classification schemes are perhaps more applicable to defining common problems in software, the taxonomies are actually difficult to represent when building genealogical structures. Because denial of service attack tools often work against several targets, from various platforms, and often exploit multiple vulnerabilities on host victims as a means of achieving the goal, they are simply too complex to classify according to conventional taxonomies. Instead attacks are classified according to the defined category requirements. As a result, these exploits force us into a definition of attack that makes classifying with independent classifiers difficult [5]. Unlike the conventional vulnerability taxonomies, attack taxonomies often maintain dependent classification mechanisms. In other words, an attack may actually be classified as being part of more than one category in the taxonomy depending upon its complexity and historical growth.

Because only three distinct categories are used as part of the classification scheme, the taxonomy proved broad enough to allow for a moderate but exhaustive means of characterizing all attacks. Perhaps more difficult though was the task of analyzing the source code accompanying the attack documentation or vulnerability description to determine its type and possible genealogical parents. Nonetheless, results proved interesting demonstrating a strong trend in implementation weakness attacks as shown in Figure 3. In fact, implementation weakness attacks are nearly doubled in the overall percentage of attacks catalogued here. Due to the complexity of modern software and hardware packages, these statistics may come of little surprise since complicated designs may often be prone to flaws or errors. Fortunately, implementation weakness attacks are perhaps the easiest attacks to prevent since many of them can either be corrected effectively by vendors or in some cases blocked by intrusion detection systems. In addition, the implementation weakness category

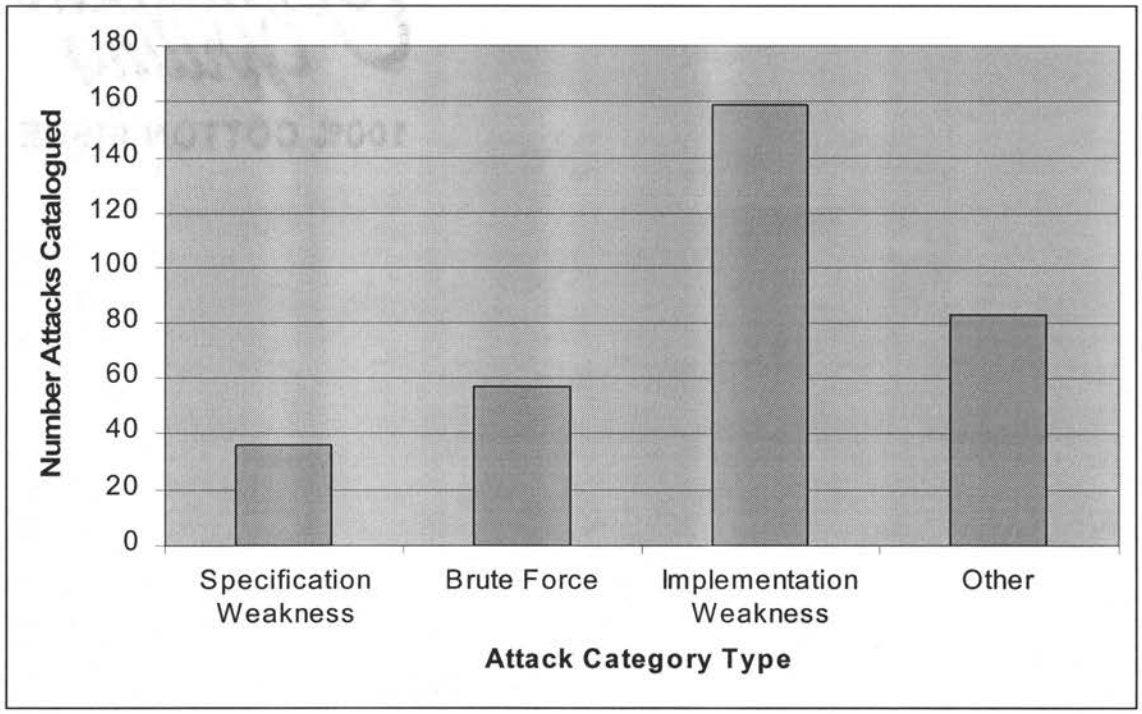

Figure 3. Attack Taxonomy Trends

also provides valuable information on how software vulnerabilities have changed over time and have attacks have been modified to overcome such changes.

Perhaps most significant from a genealogical standpoint and most interesting in attack tree structure is the miscellaneous category of dependent attack classifications. In other words, all attacks that consisted of more than one classification type placed into an undefined category then fully analyzed for its comprising types. As Figure 4 illustrates, most attacks in this category actually demonstrate all characteristics defined with the taxonomy signifying a strong growth in the complexity of the attack tools as well. In fact studying many of the attack scripts in conjunction with their respective attack trees demonstrates how many exploit tools began originally as singular implementation or specification weakness attacks and later incorporated other attack methods as well resulting many times in an increase in effectiveness and often a faster and easier means of achieving the denial of service state.

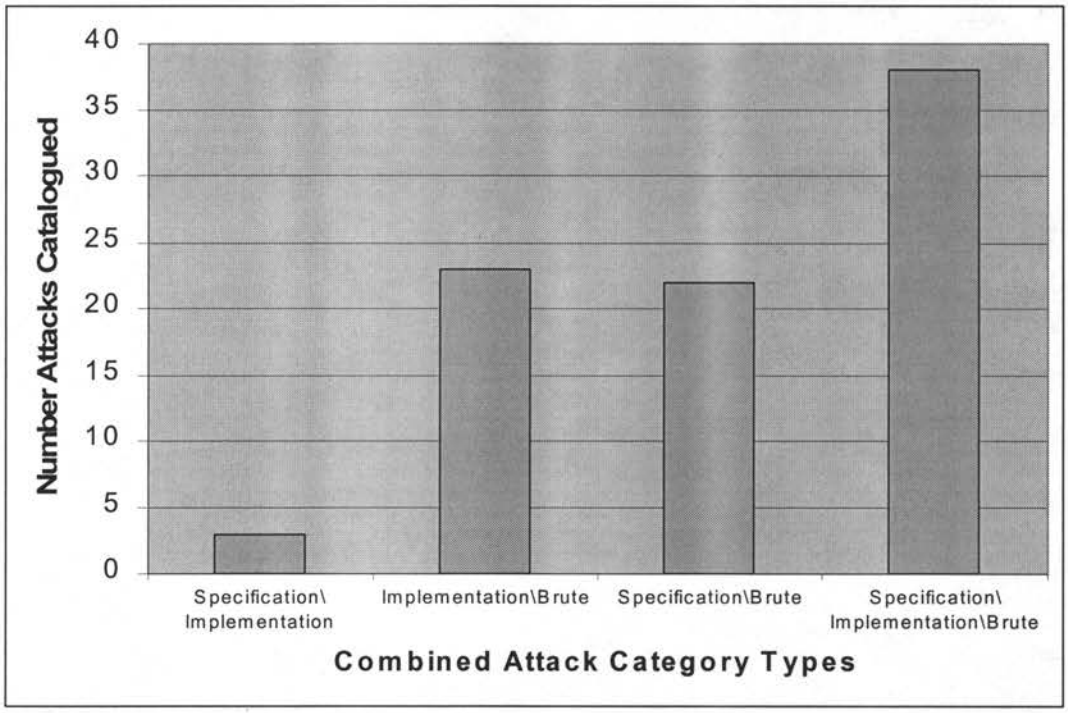

Figure 4. Continuation of Attack Taxonomy Trends

Given the subjective analysis of attack tools, the database serves an effective platform for characterizing the nature of past and existing exploit scripts for conducting denial of service. Overall, the taxonomy reveals the proportions of different attack types as well as their particular transmission methods and means of obtaining the goal. Of more concern however is how each of these attacks and attack taxonomies have changed and grown in sophistication over time.

Specification Weakness Attack Analysis

Recall that specification weakness attacks are those in which the enemy attempts directly take advantage of a flaw, weakness, or security vulnerability in the design or specification of a network, service, or communication protocol. As statistics showed in the database most often these exploits actually attack the communication protocol, or more specifically the TCP/IP protocol. Attack tools such as the classic SYN-Flood and LAND attacks specifically target weaknesses in the protocol specification. Since the vulnerability remains at the protocol specification level, the attacks are not only difficult to prevent since

doing so requires modifications of an accepted and highly implanted communication standard, but also demonstrate many of the genealogical patterns of improvement and modification important to this study.

As mentioned previously, hosts implementing the TCP/IP protocol stack are limited by the number of half-open connections allowed on any single port at some give time. Unfortunately the TCP/IP protocol requires machine resources such as computing time and memory to be devoted to each of the half-open connections. SYN-Flood attacks work by sending a targeted system a series of connection requests known as SYN packets. Although each packet causes the targeted system to issue a SYN-ACK response in acknowledgement to the connection request, the attacking system never responds to complete the three-way TCP connection handshake. While the victim waits for the ACK completion that follows the SYN-ACK response packet, the system allocates resources to the open port connection request. The resources continued to be allocated specifically to the incomplete connection process until either the attacker responds to complete the connection or and internal timer expires on the original attacker request. Since the attacker never responds to fully complete the three way TCP handshake, resources on the victim are quickly diminished. Once the limit of half-open connection requests has been reached or resources no longer become available, the system will ignore all incoming SYN requests, making the system unavailable for legitimate users.

Since TCP/IP does not incorporate strong authentication in the communication of network hosts, the attack is extremely successful in performing a remote denial of service. Furthermore, there is a requirement for an inappropriately burdensome allocation of memory and computation resources on the target side [12]. Hence for an enemy attempting to perform a successful denial of service on a system, the SYN-Flood attack approach is perhaps an extremely effective and easy means of penetration.

As a result the attack has exhibited significant improvements and variations since its debut several years ago. Originally the SYN-Flood attack script simply attempted to open as many incomplete connections on the victim as possible. The attack was effective but inefficient and easily defeated by security professionals who simply blocked multiple incoming connection requests from a single host. As the attack tree in Figure 5

demonstrates, the attack quickly evolved over the next few years into several more sophisticated attack tools that still exist as modern means of causing losses in network and resource availability.

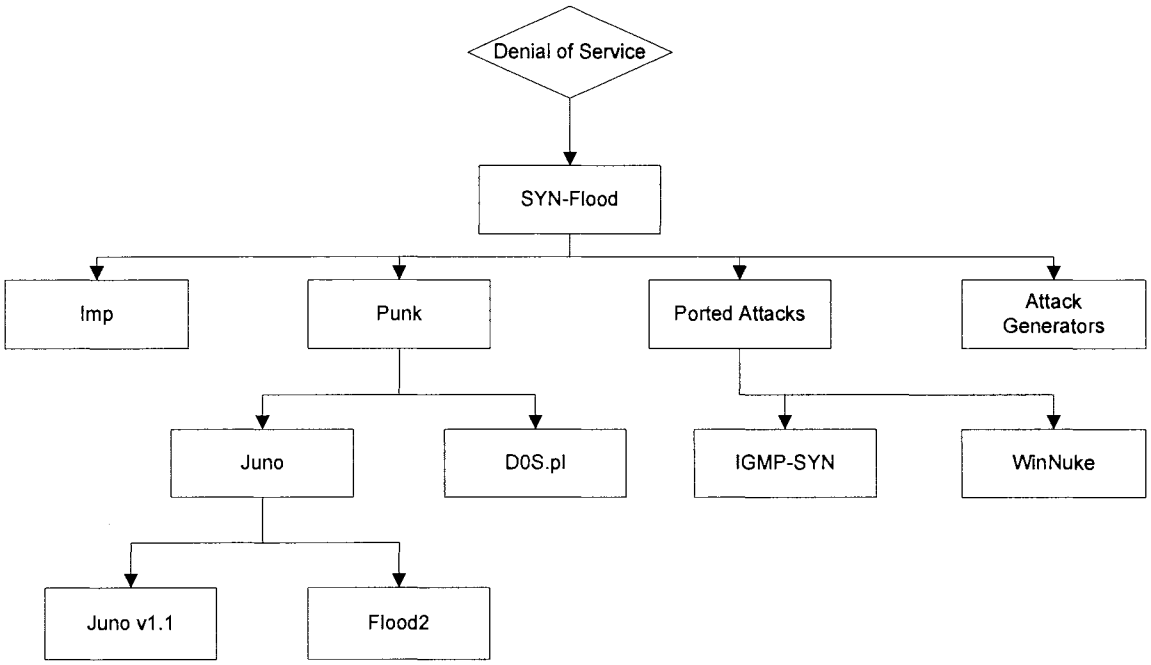

Figure 5. SYN-Flood Attack Tree Genealogy

Soon after its inception the SYN-Flood attack was modified by both the original author and several others. Shortly following the introduction of the attack, the author released the more commonly used modern release, also known as SYN-Flood 97 while others made the same simple modifications of the original attack code and released their own modified versions including Punk and Slice. The new modifications allowed for source address spoofing that make the attacks even more difficult to filter since incoming connection requests all have different forged sender addresses.

In the next year several more modifications were made to the Punk attack algorithm including porting the source code to Perl as well as revising the spoofing IP algorithm to be random instead of sequential. Once again many security experts responded by attempting to analyze forged packets and filter similar bogus connection requests. However the more significant changes to the attack came with a revision of the Punk attack script known as

Juno. The new TCP SYN-Flooding tool modified the original algorithm for faster packet creation and allowed the attacker to emulate different attacker OS platforms. In other words, the new attack tools allowed for the possibility of more rapid attacks as well as fooling filtering software into determining forged packets had been sent by separate hosts on different operating systems. Once again a short time later, Juno v1.01 was developed that allowed for even faster packet creation, better OS simulation, and improved random source IP generation [17]. In addition a separate work known as Flood2 was also introduced similar to the original Juno that conserved network bandwidth on the part of the attacker by decreasing the overall packet sizes.

The modifications and evolutionary growth of the SYN-Flood attack tree demonstrates the ability of attackers to improve designs to not only overcome additional security measures implemented by administrators but also improve the overall efficiency of attacks. Furthermore, the attack tree helps to model how similar attacks may appear in other software systems. As attack trees grow, the leaf nodes often spring entirely new methods of conducting a similar attack on either different software systems or communication networks. For example shortly after the SYN-Flood attack was released, several new attacks were developed for the Microsoft Windows platform that simply represented variations of the original SYN-Flood idea that exploits the Out-of-Band data assumption [17]. In the case of the KillWin and WinNuke attacks, the enemy sender directs a special packet to the host with the urgent pointer set. The Windows system responds by immediately allocating resources to the request. However, even though the urgent pointer is set and the Windows system expects data to follow, the attacker instead continues to follow with more Out-of-Band data connections forcing more resources to be exhausted. In fact when sent to the NETBIOS port, the service failed to even handle the requests correctly resulting in an immediate system crash.

Unfortunately, researchers may never be able to fully overcome the problems associated with specification weakness attacks since many are already inherently part of existing standards and intertwined with countless legacy network products. However studying the genealogy of attack trees such as SYN-Flood yields valuable data on how attacks have changed in the past to overcome new security mechanisms and the underlying

vulnerabilities which remain to either still be exploited or in exploited in new ways. Since the publication of the SYN-Flood vulnerability multiple similar strategies, including LAND and other TCP-Loopback attacks, have stemmed from the same attack methodology. By analyzing themes recurrent in different attack trees gain some understanding on both the mindset of attacker and the methodology of achieving denial of service.

Brute Force Attack Analysis

Recall that brute force attacks are those in which the enemy attempts to overflow or deplete computing system resources by some forceful and exhaustive means that relies on their own processing power and network bandwidth and the limitations of the system and network resources of the victim machine. Often brute force attacks simply work by flooding the victim network or system with useless data. Attacks such as Smurf, TFN2K, and Stacheldracht all act as extremely effective means of overwhelming the victim systems with vast amounts of maliciously crafted packets.

While some brute force attacks are simplistic and simply overwhelm the victim with ping requests, others demonstrate more sophistication and often will take advantage of implementation or specification weakness vulnerabilities as well. For example the Smurf attack targets a feature in the IP specification known as directed or subnet broadcasting. By definition the IP protocol specification allows for both unicast and multicast communication transmissions, meaning data packets can be either sent to a single host or group of hosts respectively. In order to send a packet to multiple machines, a machine need only to set the receiver address to the broadcast address. When a packet is sent to that IP broadcast address from a machine outside of the local network, it is broadcast to all machines on the target network (as long as routers are configured to pass along that traffic) [18]. Hence any machine is capable of sending broadcast messages to all machines on a particular local area network.

In order for the enemy to launch a Smurf attack against the network, the attacker forges simply forges ICMP echo requests using the spoofed sender addresser or the victim. However, unlike a normal ICMP echo request the receiver address is set to a broadcast address. If the routing device delivering traffic to those broadcast addresses performs the IP

broadcast function, most hosts on that IP network will take the ICMP echo request and reply to it with an echo reply each, multiplying the traffic by the number of hosts responding [19]. In other words, a ping request is sent to every host on the network with the reply address as a single victim. Once the requests are received potentially dozens of machines then begin sending replies to the target system overwhelming both the network with useless data as well as the victim's protocol stack.

Upon its release, the Smurf attack led to a denial of service of many IRC servers and their providers. Although effective, the Smurf attack demonstrated many of the same inefficiencies as other previous attacks such as the SYN-Flood. Original attacks were blocked simply by preventing IP-directed broadcasts at the router or by disabling ICMP replies on intermediary machines. However, little could be done to prevent the attack on behalf of the victim and as a result the brute force attack quickly grew in sophistication as demonstrated in Figure 6.

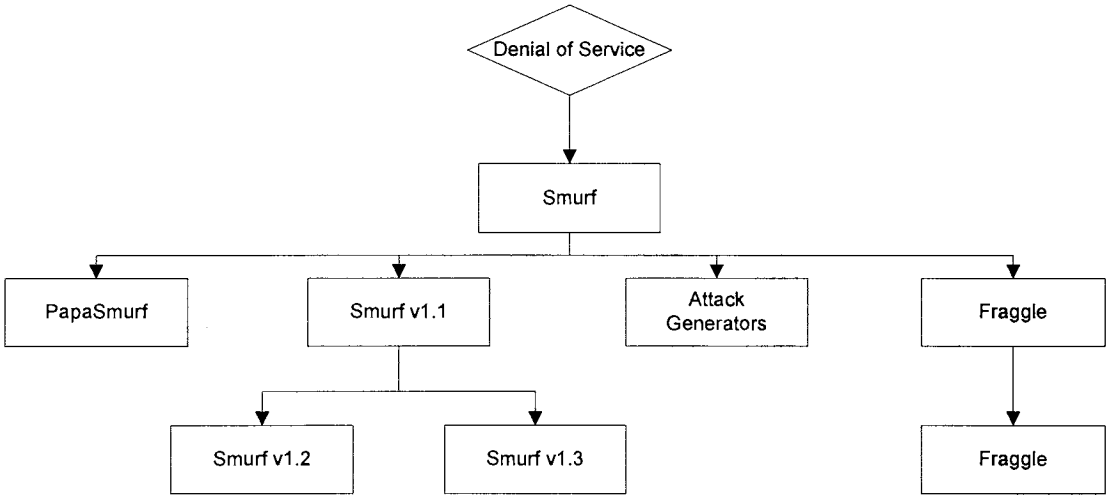

Figure 6. Smurf Attack Tree Genealogy

New revisions were released months later that demonstrated faster packet creation by interacting directly with the kernel instead of normal library routines. PapaSmurf and Rurf both modified the original attacks to grow in complexity and efficiency. However the most serious improvements came with newly revised versions of the original source code including Smurf v1.1 and v1.2. Not only did the attack increase in speed but also built in the

new capabilities as well. New attack engines were capable of scanning for vulnerable networks as well as vulnerable intermediaries capable of launching the attack against the victim. In fact, version 1.2 even built in sophisticated logging capabilities that enabled the attacker to build a comprehensive resource of vulnerable targets to help launch the denial of service. Finally, version 1.3 ported the attack to the Windows NT platform increasing the ease in which the denial of service attack could be launched by average users. No longer did attackers need access to a UNIX or Linux based platform to cause a loss of availability, nor did they require a sophisticated knowledge on how to compile and launch the attack. In just over a year, the attack had become a user-friendly tool capable of launching an extremely sophisticated attack.

While these analytical results may not appear to be entirely surprising, the research nonetheless demonstrates the ability of the attacker to improve the denial of service tools and adapt to changing security practices, such as filtering IP broadcast traffic at the router. What is more interesting though is the fact that once again the attack was ported to a different platform soon after release. Just as SYN-Flood had sparked a WinNuke attack creation, the Smurf attack was ported to the UDP protocol in a new attack called Fraggle. Much like its counterpart, the Smurf attack cousin, Fraggle, uses UDP echo packets in the same fashion as the ICMP echo packets; it was a simple re-write of Smurf [19]. Likewise, the Fraggle attack also exhibited the same revisions to improve efficiency and effectiveness soon after its release.

Much like specification weakness attacks, there is no means of overcoming brute force attacks. By their very design, computers are intended to process data and output results. By flooding the data input with useless information and overwhelming the system resources, enemies are able to launch quick and effective denial if service attacks. Although researchers may never be able to fully overcome the problems associated with brute force attacks, studying the genealogy of new attack trees may help to predict future attack patterns such as Fraggle. Analyzing the themes in attack trees and trying to predict new or missing branches plays an important role in both understanding security vulnerabilities and exploit efforts.

Implementation Weakness Attack Analysis

Recall that an implementation weakness attack is any exploitation of a hardware or software design flaw that directly threatens the availability of system resources. In other words, the attack is an exploitation of security vulnerabilities in the target system. Unlike the brute force and specification weakness attacks, implementation weakness attacks are often easy to repair since the vulnerability exists only on a certain hardware or software package. Once the design implementation is corrected in a manner that patches the vulnerability, the systems are often able to withstand the attacks. However, it is important to note that this attack category also represents the largest portion of catalogued denial of service tools. Furthermore, as the attack trees will demonstrate, the taxonomy also shows several recurring themes used by black hats to cause a loss of availability on the target systems.

For example, in later 1997, a major problem was discovered in the way many vendors had implemented the TCP/IP protocol stack. In order to accommodate, networks with different maximum date transfer unit sizes and other limitation, the TCP protocol allowed for data packets to be fragmented, or broken into smaller pieces such that each packet contain a small portion of the original data unit. When fragmented packets finally arrive to the receiving host, they are re-assembled and combined to build the original data set. However, the reassembly process implemented by several vendors expected that incoming fragments of a datagram aligned neatly in a way that data at the start of one fragment immediately follows the end of the data set in the preceding fragment. Of course normally fragmented packets do arrive neatly and reassemble as expected. However the Teardrop attack deliberately crafted packet fragments to cause reassembly problems.

Normally as new fragmented packets arrived, TCP implementations would calculate the amount of memory to allocate to the new packet by taking the difference between the end pointer of the newly arrived packet and the offset of the previous packet according to the Figure 7. Hence under normal reassembly procedures, the packets align neatly without spaces or overlaps. However, the Teardrop attack works by sending the victim specially crafted packets such that the calculated value for the new end pointer is actually less than the previous offset pointer. This can be achieved by ensuring that the second fragment specifies

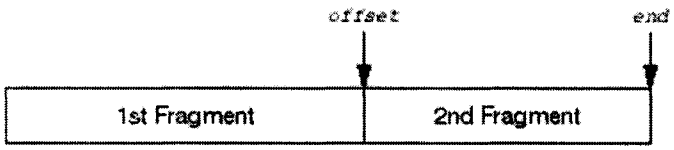

Figure 7. TCP Fragment Re-Assembly [20]

a fragment offset that resides within the data portion of the first fragment and has a length such that the end of the data carried by the second fragment is short enough to fit within the length specified by the first fragment [20]. Although the TCP specification does not allow for this problem to occur, the data packets are not properly validated by the TCP stack implementations on the victim machines.

As a result, the implementation module performing the memory allocation for the newly arrived packet attempts to copy the data into a buffer already assigned to the previous packet. Furthermore, the new total calculated length of the combined data packets actually returns a negative size value. Since the implementations expect an unsigned integer, the negative size value is actually interpreted as a very large positive integer value [20]. Upon such requests, most TCP implementations would either fail due to stack corruption or cause a complete system halt and in both cases allowing the attacker to achieve the goal state of denial of service.

Again much like the SYN-Flood and Smurf attacks, the Teardrop attack originally appeared as an attack script to launch from Linux and UNIX hosts. Unfortunately, unlike other denial of service attacks, the Teardrop could not be easily filtered by firewalls and other Internet gateways since fragments are only reassembled by the intended end receiver. Protection against the attack was reliant upon the victims to patch the appropriate software and firmware vulnerabilities on the affected machines. In the meantime as original software patches were issued by vendors to correct the problem, the Teardrop attack grew rapidly in sophistication as shown in Figure 8.

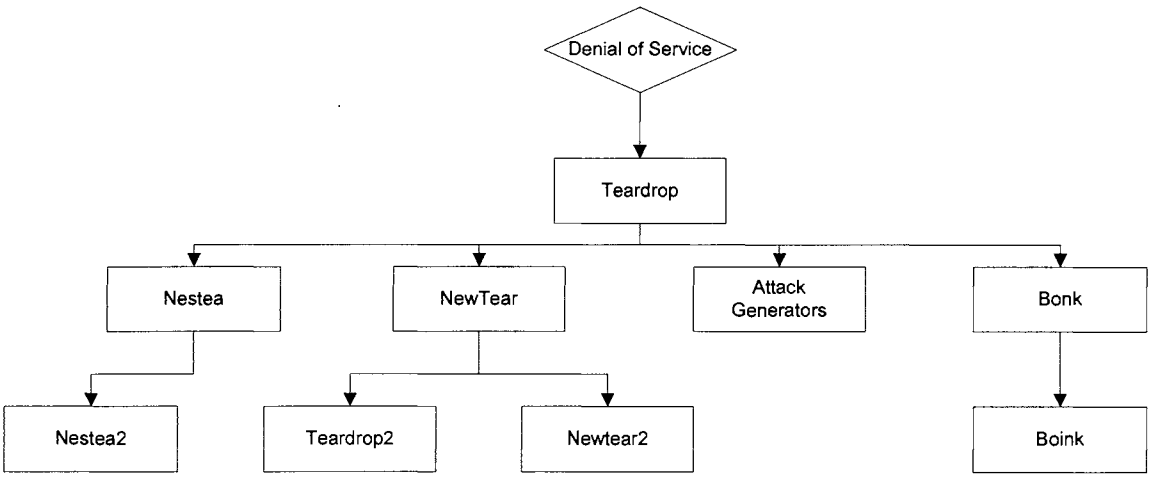

Figure 8. Teardrop Attack Tree Analysis

Originally the Teardrop attack had been coded as a proof-of-concept exploit. In other words, the original tools was a simple script to demonstrate the effectiveness of the attack as a denial of service as well as a means of testing vulnerable systems. The attack was quickly modified though to produce spoofed source address to help hide enemy identification. Eventually the attack also implemented smaller padding sizes for the first packet as well in order to reduce the overall payload size and increase the overall attack efficiency. In addition, tools such as Teardrop2 implemented faked UDP data lengths as well as a means of beginning to attack multiple protocol stacks on the victim machine as well as porting the same attack to a new platform.

Perhaps more interesting though is the Bonk branch of the Teardrop attack tree. Soon after vendors issued patches to repair the original Teardrop vulnerabilities, the black hats responded by varying the original attack slightly. Instead of setting the fragment offset to point to a memory segment already occupied by another data packet, the fragment offset is set to point to a location far beyond the end of the data packet. The Bonk attack causes the target machine to reassemble a packet that is much too big to be reassembled causing the system to crash [22]. As vendors scrambled to once again prevent the newly discovered attack, many administrators began to prevent the attacks by preventing access to the specific port the Bonk tool attempted to exploit. Shortly after, Boink was released to allow attackers

to specify port ranges, scan vulnerable hosts, and improve the speed at which malicious packets could be generated.

To make matters worse, several other IP fragmentation denial of service exploits exhibit the same themes in the attack trees. For example the Ping-of-Death attack originally published in 1996 continued to be used in several other forms, most notably Jolt and SSPing, over the course of the next few years while vendors continued to make the same mistakes in design of the ICMP fragment reassembly implementations and attackers continued to use the same methods of improving attack efficiency and effectiveness. Studying the genealogy of this tree and comparing it to other similar attacks yields valuable information on recurring patterns in software vulnerabilities as well as exploitation scenarios.

ATTACK TREE TRENDS

Although several key attack trees have been discussed individually, several more key important themes portrayed in the graphs have not been discussed. In addition to growing in sophistication and complexity, many identified attacks demonstrate other genealogical growth as well. For example most of the attacks discussed thus far were published between 1996 and 1998. However, in 1998 important new attack ideas began to surface that heavily borrowed from successful denial of service exploits to date. Both attack generators and distributed tools demonstrated the continued growth of attacks in complexity as well as hierarchically.

Growth in Sophistication

As nearly every attack taxonomy demonstrates, exploits grow in sophistication over time. In the most simplistic model attack trees, an implementation weakness is discovered and an attack is published which effectively exploits the vulnerability to cause a denial of service. After the attack is published, other malicious code authors borrow from the original idea and, more often, the source code to make improvements. These improvements typically include building a friendlier attack interface, making the attack easier to use, increasing the attack effectiveness, or in some case modifying the attack to continue to function on patched systems.

For example, the Trash denial of service exploit was originally released as a simple tool capable of generating spoofed ICMP packets with random error codes and types. Because the ICMP specification had not been correctly implemented on the Windows operating system, the spoofed ICMP packets often caused a complete system failure. A few months later after the vendor had taken the appropriate steps to repair the vulnerability, Trash2 was released which incorporated new improvements in attack efficiency as well as demonstrate the ability to now generate IGMP packets as well. Both improvements may have arguably been predicted by security researchers though the study of similar attack trees and their growth.

Likewise, exploit tools are often soon ported to either different target or attacker platforms after their initial release. For example, the implementation weakness taxonomy demonstrated a popular attack known as Teardrop. Over the course of nearly a year, the Teardrop attack spawned several other important exploits. However while the original teardrop attack focused primarily on attacking MS Windows systems, a newly developed attack known as Nestea used the same idea to attack Linux, PalmOS, HP JetDirect Cards, and several other popular platforms.

In any case, many attack trees demonstrate a growth in sophistication, complexity, as well as versatility in very short time periods. Given the common themes demonstrated specifically in the implementation weakness trees, vendors, software programmers, and engineers should not only be analyzing their own products and computing systems for undiscovered vulnerabilities to improve overall security but may also benefit from analyzing the attacks currently being used against other products as well. By studying how a specific attack is being used against one platform, researchers may indeed be able to test the attack on other systems as well and more importantly predict possible changes in the attack that may overcome the newly issued system patches.

Attack Generators

Once again in every attack category, exploits grew rapidly in sophistication. With the increases in complexity though came new denial of service tools known as attack generators. As several important implementation and specification weakness attacks were being published throughout 1996 to 1998, many malicious coders began to build tools that simply borrowed from each of the previous effective attack ideas. The new tools, often called attack generators or aggressors, once again showed a growth in complexity and sophistication but little in the way of efficiency.

New attack generator tools exhibited two important themes: combination of old attacks and randomization of new exploitation efforts. Early attack generators were often simply of conglomeration of older and previously published attacks. In other words the next logical step for attackers was to combine multiple denial of service exploits into one tool

using simple Unix shell scripts [24]. For example the popular Targa attack generator relied on eight already released attacks as shown in the attack tree in Figure 9.

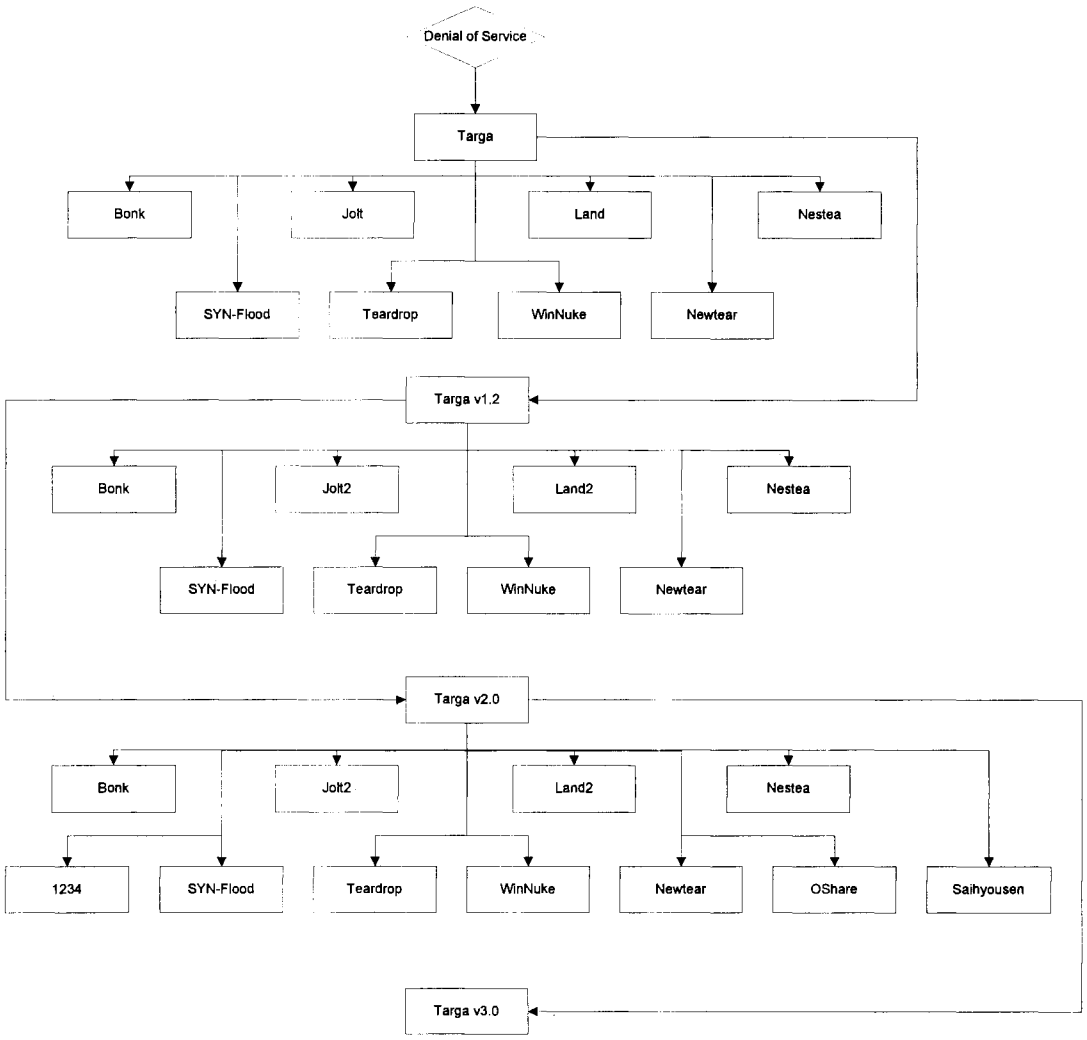

Figure 9. Targa Attack Tree

In this case the Targa attack was simply a new interface to several effective attacks, making it multi-platform and extremely effective at causing resource loss on various target systems.

Since many attackers did not completely understand the vulnerabilities that made the attacks possible nor what specific platforms the vulnerability existed on, the Targa attack probably appeared as an even more alluring tool for launching a denial of service. A tool like this has the advantage of allowing an attacker to give a single IP address and have multiple

attacks be launched increasing the probability of successful attack [24]. While this particular feature of Targa and other attack generators arguably made the new denial of service tools easier to use and more likely to be abused by those without a strong working knowledge of security, the combination of attacks did result in losses of efficiency on behalf of the attacker. Using tools such as Targa, the enemy now had to launch several completely different types of attacks against an opposing system of which few, if any, the target may have been vulnerable to.

To make matters worse for the attacker, many of the new generators began to incorporate every version of a single attack. Instead of simply including the newest revision of the SYN-Flood or Teardrop attack, even the malicious coders developing the attack generators demonstrate little knowledge on the how each individual script actually functions and what vulnerabilities it exploits. For example, DataPool was released as several different versions, the latest of which includes both Nestea and Nestea2 as well as Bonk and Boink. Hence the combination of old exploits in attack generators often leads to severe efficiency penalties as the same attacks are continuously repeated in several different fashions.

On the other hand, many attack generators exhibit an attempt on behalf of the source code author to possibly uncover new vulnerabilities and attacks by randomizing the denial of service effort. For example, another attack generator known as Aggressor as well as the forth and final release of Targa, both generate useless TCP/IP packets with random invalid fragmentation sizes, window sizes, lengths, types, and other unusual characteristics in attempt to discover unreported specification or implementation weakness attacks. Fortunately in most cases, these attack generators alone remain far from being efficient or effective.

Attack generators demonstrate both the ease of use of new exploit tools as well as their reduction in overall efficiency. Nonetheless, the models demonstrate the importance once again of recognizing vulnerabilities that have already been discovered on other platforms and ensuring the proper safeguards are implemented for any system under one's own direction. As attacks are combined and randomly launched against unsuspecting hosts, the victims become susceptible to a wide variety of possibly unpublished vulnerabilities on the target platform.

Distributed Attacks

Perhaps the most disturbing trend in the growth of attack trees is the combination, automation, and distribution of denial of service tools. Although it was not entirely possible to prevent specification weakness and brute force attacks, most Internet hosts increased in network bandwidth, processing power, as well as memory resources to a point where it became difficult for a single attacker to overwhelm the target system with enough data that it would be unable to respond to other legitimate requests. As previous attack methods began to meet their limits in less than two years, the next logical step was taken to combine the power of a number of multiple systems into a distributed denial of service cluster [24]. In 1999, several new attack tools were published that combined the strengths of both specification weakness and brute force exploits in a new coordinated and distributed denial of service interface.

By installing denial of service attack clients on multiple compromised systems and developing a hierarchy of control such that all clients could be issued commands from a central location, attackers could now launch their attack from multiple hosts. The first two tools to appear once again borrowed heavily from previously published attack tools. For example, Trinoo simply coordinated a simplistic UDP flood. By coordinating an attack by several dozen client hosts each generating literally hundreds of UDP datagrams, the attacker could now overcome the limitations of previous attacks by brute force means.

Hence many of the attack trees previously discussed spawned new nodes that demonstrated both the automation and distribution of new tools. Previous attack tools were quickly modified to incorporate attack distribution. For example, Targa was modified to create Tribe Flood Network, or TFN. Much like its Targa cousin, TFN supports ICMP flood, UDP flood, SYN-Flood, and Smurf style attacks [25]. In addition other attacks quickly demonstrated growth in sophistication once again incorporating encryption and other stealth techniques to evade capture and removal of clients.

More importantly, the attack trees now exhibited nodes that were arguably part of both the brute force and specification weakness category according to the original taxonomy. Distributed denial of service attacks signify a major change in overcoming the previous resource limitations of the attacker. Since the power of many is greater than the power of

few, coordinated and simultaneous malicious actions by some participants can always be detrimental to others if the resources of the attacker are greater than the resources of the victim [27]. New distributed attacks simply brute forced many of the previously successful attack strategies. Since intelligence and resources were no longer located on the same network, both bandwidth and computing resources were conserved on the part of the attacker.

Although the distributed attack tree branch may have been somewhat hard to predict given earlier models, many of the future branches of the distributed denial of service attack nodes may not be as necessarily as hard to predict. For example, soon after the release of Trinoo and TFN, Stacheldracht and TFN2K appeared incorporating new methods of encrypting client and attacker communications to ensure that information remain hidden and to prevent normal session hijacking. In addition both attacks modified the original attack algorithms once again for a slight improvement in efficiency. Just as previous attack trees demonstrated incorporation of other attacks, scanning tools, and other features, the new distributed attack tools may soon begin to exhibit the same characteristics as well.

COUNTERMEASURES AND FUTURE WORK

While several measures currently exist as means of helping to mitigate the effects of denial of service attacks including load balancing, resource throttling, attack detection, containment, and filtering, there currently exists no full solution for entirely protecting a system against the threat of loss of availability. For example, CERT and several security vendors all advise several appropriate measures a vulnerable host can take in order to help prepare and withstand such an attack. However, given the specification weaknesses in widely used communication protocols such as TCP/IP and the inherent vulnerabilities of any host to a brute force attack, no formidable solution may exist in the near future that will entire protect systems from denial of service attacks.

In computer and network security, very few solutions actually exist though that entirely prevent many types of attacks. Instead security is an ongoing process of risk assessment. Improving the security of a system requires the users to determine where the current vulnerabilities exist, which are most likely to be exploited, and which represent the greatest risk. Fortunately, attack trees provide a formal methodology for analyzing the security of systems and subsystems [16]. In other words, attack trees form a groundwork for understanding the process of improving security.

Typically attack graphs are used by security professionals in order to determine every possible means of penetrating a vulnerable system. By trying to determine every method of obtaining the attack goal, security engineers and network administrators can they determine which nodes in the tree have the least cost to the attacker in terms of both time and dollars, require the least skill on behalf of the attacker, and are the most easily accomplished. By sending appropriate values to each node, administrators can often determine likely paths of system penetration and the possible attack scenarios for an enemy to successfully reach the final goal.

Much like their attack graph counterparts, genealogical attack trees demonstrate possible scenarios for achieving some type of attack such as a denial of service. More importantly though, attack trees represent the path that malicious users have already taken in order to successfully achieve a goal. The genealogical hierarchy of an attack tree portrays not only how attacks have changed over time but also how they have *not* changed. As

demonstrated by the Teardrop attack tree, several important paths of attacking vulnerable implementations of the TCP/IP protocol stack existed. However, at the time the initial software patches were released only one particular method of attack was currently being used by the enemy. Soon afterwards as the attack started to become ineffective against many repaired target machines, a new branch of the attack tree spawned which exploited the same fragmentation vulnerability as before by simply another means.

Studying the genealogy of the fragmentation exploits and other implementation weakness attack trees, software designers can potentially determine what other undiscovered vulnerabilities may exist. For example, had software designers fully analyzed the current attack tree structure of the Teardrop attack they may indeed have noticed other branches within the attack tree that were currently missing as part of the hierarchy. In other words, existing attack trees can be analyzed by software engineers as a methodical approach of attempting to determine other future methods of attack, likelihood of exploitation, and possible new attack trends. By analyzing the current structure and genealogy of the tree and attempting to determine currently missing branches, designers gain a valuable tool in helping to prevent future vulnerabilities. Just as administrators and security engineers use attack graphs as a methodical approach to representing important risks currently exhibited by a computing system, attack trees can be used by software designers and patch developers to help determine new attack trends and software vulnerabilities.

Software vulnerabilities exist because it is impossible for designers to produce systems which conform to extremely strict security standards without formal proofs and validation of the source code. Although many designers take careful steps to prevent software vulnerabilities, vendors simply can not predict what flaws will later be discovered. Thus attack trees became an extremely important mechanism for building more secure software. Should designers analyze the current attack against the system using a more methodical approach and determine other likely methods of attack as well as existing attack improvements, they may indeed uncover other vulnerabilities that would otherwise not be noted until later.

In addition, attack trees provide an effective means for other vendors to determine whether their products are currently vulnerable to attacks being launched on other platforms. Several of the attack trees demonstrated that attackers later ported the attack to a wide variety of other platforms each of which was also susceptible to risk. Attack trees provide a way to think about security, to capture and reuse expertise about security, and to respond to changes in security [16]. Given many of the common themes demonstrated in the implementation weakness trees, engineers may be able to gain valuable information on undiscovered vulnerabilities in their own software designs simply by analyzing the attacks currently being used against other products. By studying how a specific attack is being leveraged against one particular software application, researchers may indeed be able to test the attack on other systems as well. More importantly researchers may gain the ability to predict possible changes in the attack that could possibly overcome the newly issued system patches.

CONCLUSION

As the connectivity of modern computers continues to increase and Internet infrastructure continues to expand, the importance of maintaining the availability of network and computing resources has become an extremely important and challenging task. Recent distributed denial of service attacks demonstrate the serious vulnerabilities that still remain within much of the world's electronic communication system and the crippling effects of simple to sophisticated attack agents. Although no absolute countermeasure against the attacks currently exists, analyzing the patterns, themes, and genealogical growth of current attacks yields valuable information on existing software vulnerabilities and possible future attack trends.

By examining denial of service attack history and taxonomy together, researchers gain the ability to not only identify existing attacks and possible partial countermeasures but possibly even predict future attacks in some cases as well. Although attacks have grown increasingly complex over time, many of the same basic ideas and methods for performing the attack remain unchanged or only slightly modified. Many of the same attack themes that have been uncovered in the past are reappearing in new forms which target different software platforms, improve attack efficiency, or distribute the attack among multiple hosts. By studying the genealogy of existing attack tools and their hierarchical structure, researchers are afforded the ability to study how software vulnerability exploits have changed and migrated over time. Building a software vulnerability database of denial of service attacks comprised of both singular entries and corresponding attack trees allows researchers to refine attack taxonomies, determine existing attack patterns, and even predict future changes aimed at preventing some denial of service attacks.

BIBLIOGRAPHY

[1] Garfinkel, S. and Spafford, E. (1991). *Practical UNIX Security*. O'Reilly and Associates.

[2] Russell, D. and Gangemi, G.T. (1991). *Computer Security Basics*. O'Reilly and Associates.

[3] Arbaugh, W.A., Shankar, N, and Wan, Y.C.J. (2001). *Your 802.11b Wireless Network has No Clothes*. University of Maryland.

[4] Kumar, S. (1995). *Classification and Detection of Computer Intrusions*, Ph.D. Thesis. Purdue University.

[5] Mell, Peter. (1999). *Understanding the Global Attack Toolkit Using a Database of Dependent Classifiers*. Proceedings on the 2nd Annual Workshop on Research with Security Vulnerability Databases.

[6] Krsul, I. (1998). *Software Vulnerability Analysis*, Ph.D. Thesis. Purdue University.

[7] Moore, D., Voelker, G.M., and Savage, S. (2001) *Inferring Internet Denial-of-Service Activity*. Proceedings of the 10th USENIX Security Symposium. USENIX.

[8] Corcoran, E. (1996). *Hackers strike at N.Y. Internet Access Company*. The Washington Post, Sep. 12, 1996.

[9] CERT Coordination Center. (1997). *Denial of Service Attacks*. Pittsburgh, PA, Carnegie Mellon Software Engineering Institute.

[10] Giovanni, C. (2000) *Topology of a Denial of Service*. Endeavor Systems.

[11] CERT Coordination Center. (2001). *Managing the Threat of Denial of Service Attacks*. Pittsburgh, PA, Carnegie Mellon Software Engineering Institute.

[12] Schuba, C., Krsul, I., Kuhn, M., Spafford, G., Sundaram, A., and Zamboni, D. (1997). *Analysis of a Denial of Service Attack on TCP*. Proceedings of the 1997 IEEE Symposium on Security and Privacy.

[13] Richardson, T.W. (2001). *The Development of a Database Taxonomy of Vulnerabilites to Support the Study of Denial of Service Attacks*, Ph.D. Thesis. Iowa State University.

[14] Goldberg, I., Borisov, N., and Wagner, D. (2001) *(In)Security of the WEP Algorithm.* [On-line]. Available: http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html. (Date Accessed: 19 July 2002).

[15] Sheyner, O. Haines, J., Somesh, J., Lippman, R., and Wing, J.M. (2002). *Automated Generation and Analysis of Attack Graphs*. Defense Advanced Research Projects. [On-line]. Available: http://www-2.cs.cmu.edu/afs/cs/project/calder/papers/sp02/paper.ps. (Date Accessed: 19 July 2002).

[16] Schneier, B. (1999). *Attack Trees: Modeling Security Threats*. Dr. Dobb's Journal. [On-line]. Available: http://www.ddj.com/documents/s=896/ddj9912a/9912a.htm. (Date Accessed: 19 July 2002).

[17] Packetstorm Security. (2001). *Denial of Service Attack Tools.* [On-line]. Available: http://www.packetstormsecurity.com. (Date Accessed: 19 July 2002).

[18] CERT Coordination Center. (1998). *Smurf IP Denial of Service Attacks*. Pittsburgh, PA, Carnegie Mellon Software Engineering Institute.

[19] Huegen, C. (1998). Smurf Attack. [On-Line]. Available: http://www.quadrunner.com/~chuegen/smurf.txt. (Date Accessed: 19 July 2002).

[20] Hoggan, David. (1994). *The Internet Book.* Hoggan Website. [On-line]. Available: http://www.theinternetbook.net/. (Date Accessed: 19 July 2002).

[21] Schuba, C., McMillan, R., Garfinkel, S., Cohen, F., and Ko, H.P. (1996). *UC Davis Vulnerabilities Project: New Attacks and New Twists on Existing Attacks.* NISSC Panel on Vulnerabilities Data. [On-line]. Available: http://seclab.cs.ucdavis.edu. (Date Accessed: 19 July 2002).

[22] WWDSI. (2001). Saint Tutorial: Boink Attack. [On-line]. Available: http://www.wwdsi.com/demo/saint_tutorials/boink.html. (Date Accessed: 19 July 2002).

[23] Spatscheck, O., and Peterson, L. (1999). *Defending Against Denial of Service Attacks in Scout.* Proceedings of the 3rd Symposium on Operating Systems Design and Implementation. USENIX.

[24] Dittrich, D. (1999). *The DoS Project's "Trinoo" Distributed Denial of Service Attack Tool.* Seattle, WA, University of Washington.

[25] Dittrich, D. (1999). *The "Tribe Flood Network" Distributed Denial of Service Attack Tool.* Seattle, WA, University of Washington.

[26] Fulp, E., Fu, Z., Reeves, D.S., Wu, S.F., Zhang, X. (2001). *Preventing Denial of Service Attacks on Quality of Service.* DARPA Information Survivability Conference and Exposition II.

[27] Mirkovic, J., Martin, J., and Reiher, P. (2001). A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. Los Angeles, CA, University of California Computer Science Department.

[28] Williams, J. (1999). *Using a Database of Threats and Countermeasures to Build Protection Profiles.* Proceedings of the NIAP Workshop Held at the National Institute of Standards and Technology (NIST).

[29] SANS Institute. (2000). *Consensus Roadmap for Defeating Distributed Denial of Service Attacks.* Project for the Partnership for Critical Infrastructure Security.

[30] Dittrich, D. (2000). *Distributed Denial of Service Attack Tool Timeline.* Proceedings of the 9th Annual USENIX Security Symposium.